题目要求：
1. 利用 PLY 实现简单的 Python 程序的解析

（1）示例程序位于 example/

（2）需要进行解析的文件为 example.py

（3）需要完成以下内容的解析
- 赋值语句
- 完整的四则运算
- print 语句

四则运算的无二义性下文法大致如下：
```
expr -> expr + term | term
term -> term * factor | factor
factor -> id | (expr)
```
（不需要消除二义性）

（4）解析结果以语法树的形式呈现

example.py 文件内容如下：

```
a = 1
b = 2
c = a + b
d = c - 1 + a
print(c)
print(a, b, c)
```

程序说明：

1. 打开 main.py 文件，确保 source 中的所有代码在同一目录下

2. 确保已经安装了 PLY 库

3. 运行 main.py 文件

4. 对 example.py 文件中的程序段进行解析，结果以语法树的形式展现，并展示 print 的结果以及所有变量的最终值字典，解析结果如下图：

```
+ [PROGRAM]
  + [STATEMENTS]
    + [STATEMENTS]
      + [STATEMENTS]
        + [STATEMENTS]
          + [STATEMENTS]
            + [STATEMENTS]
              + [STATEMENTS]
                + ['STATEMENT']
                  + [ASSIGNMENT]
                    + a
                    + =
                    + 1
              + ['STATEMENT']
                + [ASSIGNMENT]
                  + b
                  + =
                  + 2
            + ['STATEMENT']
              + [OPERATION]
                + c
                + =
                + [EXPR]
                  + [EXPR]
                    + [TERM]
                      + [FACTOR]
                        + a
                  + +
                  + [TERM]
                    + [FACTOR]
                      + b
          + ['STATEMENT']
            + [OPERATION]
              + d
              + =
              + [EXPR]
                + [EXPR]
                  + [EXPR]
                    + [TERM]
                      + [FACTOR]
                        + c
                  + -
                  + [TERM]
                    + [FACTOR]
                      + 1
                + +
                + [TERM]
                  + [FACTOR]
                    + a
        + ['STATEMENT']
          + [PRINT]
            + print
            + (
            + [SENTENCE]
              + [WORD]
                + c
            + )
      + ['STATEMENT']
        + [PRINT]
          + print
          + (
          + [SENTENCE]
            + [WORD]
              + a
            + ,
            + [SENTENCE]
              + [WORD]
                + b
              + ,
              + [SENTENCE]
                + [WORD]
                  + c
          + )
```

```
3.0
1.0 , 2.0 , 3.0
{'a': 1.0, 'b': 2.0, 'c': 3.0, 'd': 3.0}
```

（第一张图是语法树的结构，具体的结点含义后文讲解。第二张图是 print 的结果，也就是 c 的最终值，还有 a, b, c 的最终值，以及所有变量的最终值）

显然解析结果和题目中需要完成的测试结果等价。

5. 对 yacc 程序定义的文法规则的解释

总体结构和 assignment 部分不多作赘述，结构较为简单，如下图：

```
def p_program(t):
    '''program : statements'''
    if len(t) == 2:
        t[0] = node('[PROGRAM]')
        t[0].add(t[1])


def p_statements(t):
    '''statements : statements statement
                  | statement'''
    if len(t) == 3:
        t[0] = node('[STATEMENTS]')
        t[0].add(t[1])
        t[0].add(t[2])
    elif len(t) == 2:
        t[0] = node('[STATEMENTS]')
        t[0].add(t[1])


def p_statement(t):
    ''' statement : assignment
                  | operation
                  | print'''
    if len(t) == 2:
        t[0] = node(['STATEMENT'])
        t[0].add(t[1])


def p_assignment(t):
    '''assignment : VARIABLE '=' NUMBER'''
    if len(t) == 4:
        t[0] = node('[ASSIGNMENT]')
        t[0].add(node(t[1]))
        t[0].add(node(t[2]))
        t[0].add(num_node(t[3]))
```

Operation 部分的文法结构如下：

文法：

Operation -> VARIABLE = expression
Expression -> expression + term | expression – term | term
Term -> term *factor | term / factor | factor
Factor -> NUMBER | VARIABLE | (expression)

```
def p_operation(t):
    '''operation : VARIABLE '=' expression'''
    if len(t) == 4:
        t[0] = node('[OPERATION]')
        t[0].add(node(t[1]))
        t[0].add(node(t[2]))
        t[0].add(t[3])


def p_expression(t):
    '''expression : expression '+' term
                  | expression '-' term
                  | term'''
    if len(t) == 4:
        t[0] = node('[EXPR]')
        t[0].add(t[1])
        t[0].add(node(t[2]))
        t[0].add(t[3])
    elif len(t) == 2:
        t[0] = node('[EXPR]')
        t[0].add(t[1])


def p_term(t):
    ''' term : term '*' factor
             | term '/' factor
             | factor'''
    if len(t) == 4:
        t[0] = node('[TERM]')
        t[0].add(t[1])
        t[0].add(node(t[2]))
        t[0].add(t[3])
    elif len(t) == 2:
        t[0] = node('[TERM]')
        t[0].add(t[1])


def p_factor(t):
    '''factor : VARIABLE
              | '(' expression ')'
              | NUMBER'''
    if len(t) == 2:
        t[0] = node('[FACTOR]')
        try:
            if type(eval(t[1])) == int or float:
                t[0].add(num_node(t[1]))
        except NameError:
            t[0].add(node(t[1]))
    elif len(t) == 4:
        t[0] = node('[FACTOR]')
        t[0].add(node(t[1]))
        t[0].add(t[2])
        t[0].add(node(t[3]))
```

Print 部分的文法结构如下图:

文法:

Print -> PRINT (sentence)
Sentence -> word, sentence | word
Word -> NUMBER | VARIABLE

```
def p_print(t):
    '''print : PRINT '(' sentence ')' '''
    t[0] = node('[PRINT]')
    t[0].add(node(t[1]))
    t[0].add(node(t[2]))
    t[0].add(t[3])
    t[0].add(node(t[4]))


def p_sentence(t):
    '''sentence : word ',' sentence
                | word'''
    t[0] = node('[SENTENCE]')
    if len(t) == 2:
        t[0].add(t[1])
    else:
        t[0].add(t[1])
        t[0].add(node(t[2]))
        t[0].add(t[3])


def p_word(t):
    '''word : NUMBER
            | VARIABLE'''
    t[0] = node('[WORD]')
    try:
        if type(eval(t[1])) == (int or float):
            t[0].add(num_node(t[1]))
    except NameError:
        t[0].add(node(t[1]))
```

6. Translation 部分

理解各层的 node 里面应该是什么值就行，比如说 operation 部分各层的 node 的 value 就是当前的运算结果，而 print 部分 sentence 层的 node 的 value 就是待输出的值组成的列表，然后对各层执行相应的 setvalue，更新 v_table 等操作即可。